Nombre: Daniel Andrés Vasquez Murillo
Código de estudiante: 8963154

Tarea 1

Problemas conceptuales

1. Escribir el código de honor del curso.

   Como miembro de la comunidad académica de la Pontificia Universidad Javeriana Cali me comprometo a seguir los más altos estándares de integridad académica.

2. Problema 1-1: Comparison of running times (Cormen et al., página 15).

1-1 *Comparison of running times*

For each function $f(n)$ and time t in the following table, determine the largest size n of a problem that can be solved in time t, assuming that the algorithm to solve the problem takes $f(n)$ microseconds.

	1 second	1 minute	1 hour	1 day	1 month	1 year	1 century
$\lg n$							
$\sqrt{n}$							
n							
$n \lg n$							
n^2							
n^3							
2^n							
$n!$							

Lo primero que se hace es tomar las conversiones a microsegundos de cada uno de los tiempos:

- 1 segundo = 10^6
- 1 minuto = $60 * 10^6$
- 1 hora = $36 * 10^8$
- 1 día = $864 * 10^8$
- 1 mes = $26298 * 10^8$
- 1 año = $315576 * 10^8$
- 1 siglo = $315576 * 10^{10}$

Lo siguiente es encontrar si es posible la ecuación para encontrar el valor de "n" cuando la función "f(n)" dura los anteriores tiempos tomados en sus respectivas en microsegundos, Por ejemplo, para cuando f(n) = n entonces los tiempos no se ven cambiados en nada y la tabla queda de la siguiente forma:

	1 segundo	1 minuto	1 hora	1 día	1 mes	1 año	1 siglo
$\lg n$							
$\sqrt{n}$							
n	10^6	$60 * 10^6$	$36 * 10^8$	$864 * 10^8$	$26298 * 10^8$	$315576 * 10^8$	$315576 * 10^{10}$
$n \lg n$							
n^2							
n^3							
2^n							
$n!$							

Ahora para los otros f(n) la forma en la que encontré sus n es la siguiente:

1. Para $\mathbf{lg}\,\boldsymbol{n}$

   Asumiendo que log es de base 10 entonces n = 10^t

2. Para $\sqrt{\boldsymbol{n}}$ = t

   n = t^2

3. Para n
   n = t
4. Para n * lg n = t
   Debido a no poder encontrar una ecuación para resolverlo, recurrí a utilizar un código en Python con el cual fui buscando iterativamente el valor de n sumando n + 1, n + 100, n + 1000, n + 10000, n + 1000000, durante el ciclo a medida que el valor de t aumentaba de tamaño, para que la ejecución del código no sea muy lenta con los números cada vez más grandes. El código es el siguiente:

```python
import math

t_1 = 10**6
t_2 = 60 * 10**6
t_3 = 36 * 10**8
t_4 = 864 * 10**8
t_5 = 26298 * 10**8
t_6 = 315576 * 10**8
t_7 = 315576 * 10**10


def encontrar_n(n,t):
    while math.log(n, 10) * n < t:
        n += 1

    return n - 1

n = 1
print("1 segundo =", encontrar_n(n,t_1))

n = 1
while math.log(n, 10) * n < t_2 - 100:
        n += 100
print("1 minuto =", encontrar_n(n,t_2))

n = 1
while math.log(n, 10) * n < t_3 - 100:
        n += 100
print("1 hora =", encontrar_n(n,t_3))

n = 1
while math.log(n, 10) * n < t_4 - 1000:
        n += 1000
print("1 dia =", encontrar_n(n,t_4))
```

```python
n = 1
while math.log(n, 10) * n < t_5 - 10000:
        n += 10000
print("1 mes =", encontrar_n(n,t_5))

n = 1
while math.log(n, 10) * n < t_6 - 100000:
        n += 100000
print("1 año =", encontrar_n(n,t_6))

n = 1
while math.log(n, 10) * n < t_7 - 1000000:
        n += 1000000
print("1 siglo =", encontrar_n(n,t_7))
```

```
1 segundo = 189481
1 minuto = 8649300
1 hora = 417596800
1 dia = 8692885000
1 mes = 231407320000
1 año = 2543841500000
1 siglo = 220028898000000
```

5. Para n^2 = t
   n = $\sqrt[2]{t}$
6. Para n^3 = t
   n = $\sqrt[3]{t}$
7. Para 2^n = t
   Manteniendo la igualdad aplico logaritmo $lg(2^n)$ = lg t
   Aplicando leyes de los logaritmos $n * lg(2)$ = lg t
   Entonces n = lg(t) / $lg(2)$
8. Para n!
   Debido a no poder encontrar una ecuación para resolverlo, entonces fui sumando de 1 en 1 a "n" y aplicando la factorial capturando el n anterior al que se pasara en cada tiempo.
   También una forma de hacerlo en código es la siguiente:

```python
tiempos = [10**6, 60 * 10**6, 36 * 10**8, 864 * 10**8, 26298 * 10**8, 315576 * 10**8, 315576 * 10**10]

duraciones = ["1 segundo", "1 minuto", "1 hora", "1 dia", "1 mes", "1 año", "1 siglo"]

duracion = 0
for i in tiempos:
    n = 1
    factorial = 1
    while factorial < i:
        factorial += factorial * n
        n += 1
    print(duraciones[duracion], n - 1)
    duracion += 1
```

```
1 segundo 9
1 minuto 11
1 hora 12
1 dia 13
1 mes 15
1 año 16
1 siglo 17
```

Se han descartado los números decimales de los resultados para manejar únicamente enteros.

	1segundo	1 minuto	1 hora	1 día	1 mes	1 año	1 siglo
$\lg n$	10^{10^6}	10^{60*10^6}	10^{36*10^8}	10^{864*10^8}	$10^{26298*10^8}$	$10^{315576*10^8}$	$10^{315576*10^{10}}$
$\sqrt{n}$	10^{12}	$60^2 * 10^{12}$	$1296^2 * 10^{16}$	$864 * 10^{16}$	$26298^2 * 10^{16}$	$315576^2 * 10^{16}$	$315576^2 * 10^{20}$
n	10^6	$60 * 10^6$	$36 * 10^8$	$864 * 10^8$	$26298 * 10^8$	$315576 * 10^8$	$315576 * 10^{10}$
$n \lg n$	189481	8649300	417596800	8692885000	231407320000	2543841500000	220028898000000
n^2	1000	7745	60000	293938	1621665	5617615	56176151
n^3	100	391	1532	4420	13803	31601	146679
2^n	19	25	31	36	41	44	51
$n!$	9	11	12	13	15	16	17

Problemas prácticos

Colaboración

Se analizaron los enunciados de cada punto del A al E con los compañeros Miguel Ángel Nivia y Miguel Ángel Sánchez. Exclusivamente para entender las dinámicas a aplicar mas no para sacar un código solución de cada punto.

Bibliografía

- Se utilizo únicamente para entender el Segundo punto conceptual CLRS Solutions | Problem 1-1 | The Role of Algorithms in Computing (atekihcan.github.io)
- Para facilitar los cálculos en el problema 1-1 se utilizó Symbolab Math Solver - Step by Step calculator